L Number	Hits	Search Text	DB	Time stamp
-	246	5,206,939	USPAT	2003/12/10 13:46
-	1163	(data adj storage adj system) and (director controller) and (disk adj drive)	USPAT	2003/12/11 14:08
-	330	((data adj storage adj system) and (director controller) and (disk adj drive)) and (intercommunication messag$3)	USPAT	2003/12/11 14:08
-	330	(((data adj storage adj system) and (director controller) and (disk adj drive)) and (intercommunication messag$3)) not enc	USPAT	2003/12/11 10:16
-	250	(((data adj storage adj system) and (director controller) and (disk adj drive)) and (intercommunication messag$3)) not emc	USPAT	2003/12/12 10:51
-	51	((data adj storage adj system) and (director controller) and (disk adj drive)) and crossbar	USPAT	2003/12/11 10:24
-	37	(((data adj storage adj system) and (director controller) and (disk adj drive)) and crossbar) not emc	USPAT	2003/12/11 10:24
-	29	(data adj storage adj system) and (director controller) and (disk adj drive) and (message near (center system unit))	USPAT	2003/12/11 13:52
-	40	(data adj storage adj system) and (director controller) and (disk adj drive) and (messag$3 near (center system unit))	USPAT	2003/12/11 14:04
-	20	(data adj storage adj system) and (director controller) and (disk adj drive) and (messag$3 near network)	USPAT	2003/12/11 14:04
-	1	((data adj storage adj system) and (director controller) and (disk adj drive)) and (intercommunication)	USPAT	2003/12/11 14:16
-	5	6,272,662	USPAT	2003/12/11 14:14
-	323	((data adj storage adj system) and (director controller) and (disk adj drive)) and (messag$3) and (verif$4 verification determin$3)	USPAT	2003/12/11 14:17

US006061274A

United States Patent [19]

Thibault et al.

[11] Patent Number: 6,061,274

[45] Date of Patent: May 9, 2000

[54] METHODS AND APPARATUS FOR MESSAGE TRANSFER IN COMPUTER STORAGE SYSTEM

[75] Inventors: **Robert A. Thibault**, Raynham; **Michael Shulman**, Groton, both of Mass.

[73] Assignee: **EMC Corporation**, Hopkinton, Mass.

[21] Appl. No.: **09/224,701**

[22] Filed: **Jan. 4, 1999**

[51] **Int. Cl.**[7] **G11C 7/00**

[52] **U.S. Cl.** **365/189.05**; 365/241

[58] **Field of Search** 365/189.01, 189.02, 365/189.04, 189.05, 241, 230.02, 230.03

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,119,240 6/1992 Raymond 365/230.03

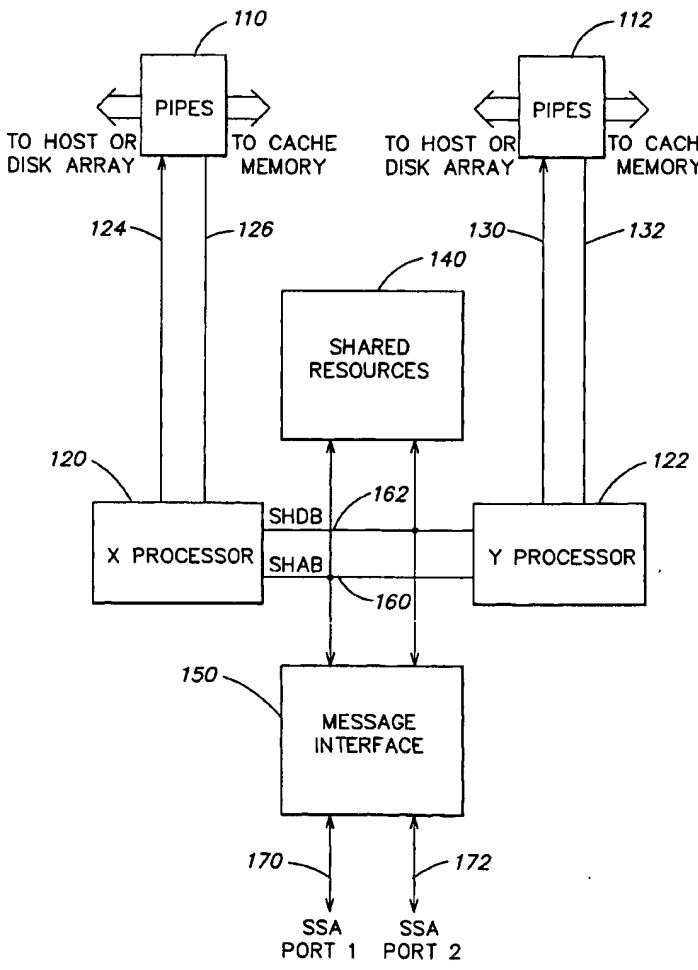
Primary Examiner—Terrell W. Fears

Attorney, Agent, or Firm—Wolf, Greenfield & Sacks, P.C.

[57] **ABSTRACT**

A computer storage system includes an array of storage devices, a system cache memory, one or more back end directors for controlling data transfer between the storage devices and the system cache memory, and one or more front end directors for controlling data transfer between the system cache memory and a host computer. Each director includes a processor and a message interface for controlling high speed message transfer between the processors in the directors. The message interfaces in the directors may be interconnected in a closed-loop configuration. Each message interface may include a transmit/receive circuit for transferring messages to and from other directors, a message memory for storing outgoing messages and incoming messages, and a message controller for controlling transfer of messages between the processor and the message memory and between the message memory and the transmit/receive circuit.

18 Claims, 7 Drawing Sheets



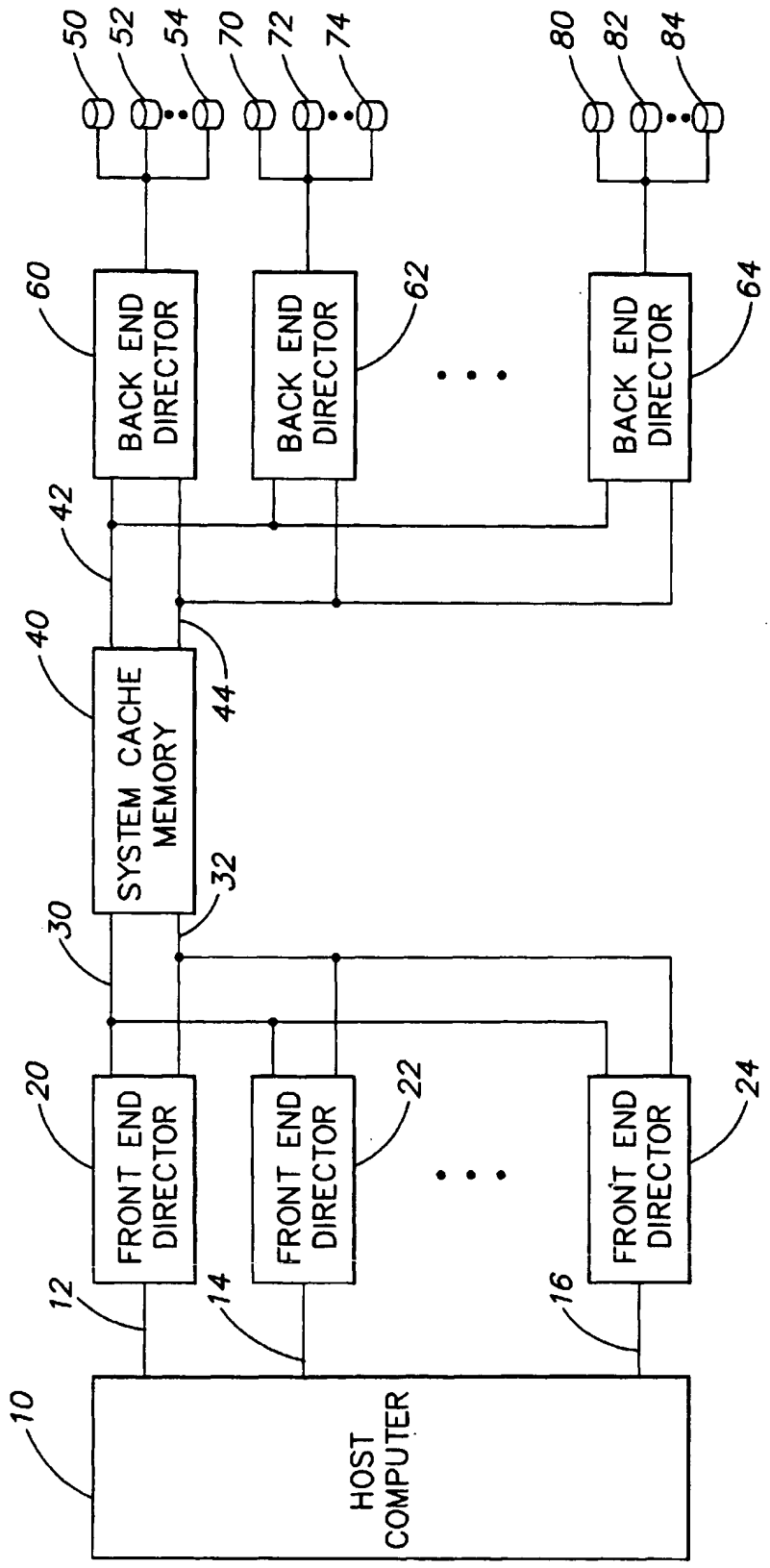

50
52
54
70
72
74
80
82
84
60
BACK END DIRECTOR
62
BACK END DIRECTOR
64
BACK END DIRECTOR
42
40
SYSTEM CACHE MEMORY
44
32
30
20
FRONT END DIRECTOR
22
FRONT END DIRECTOR
24
FRONT END DIRECTOR
12
14
16
10
HOST COMPUTER

FIG. 1

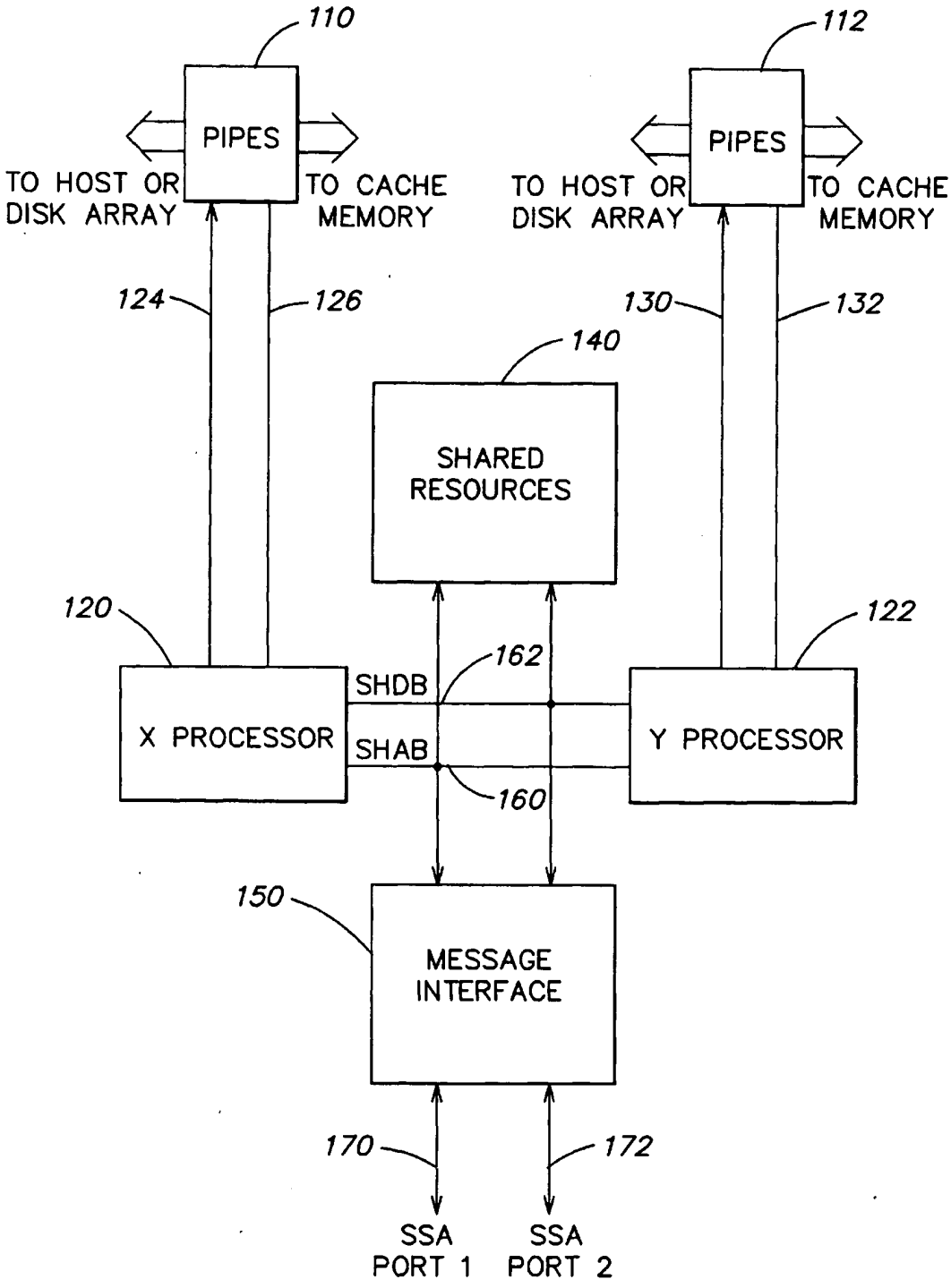

110
112
PIPES
PIPES
TO HOST OR DISK ARRAY
TO CACHE MEMORY
TO HOST OR DISK ARRAY
TO CACHE MEMORY
124
126
130
132
140
SHARED RESOURCES
120
122
162
SHDB
SHAB
X PROCESSOR
Y PROCESSOR
160
150
MESSAGE INTERFACE
170
172
SSA PORT 1
SSA PORT 2

FIG. 2

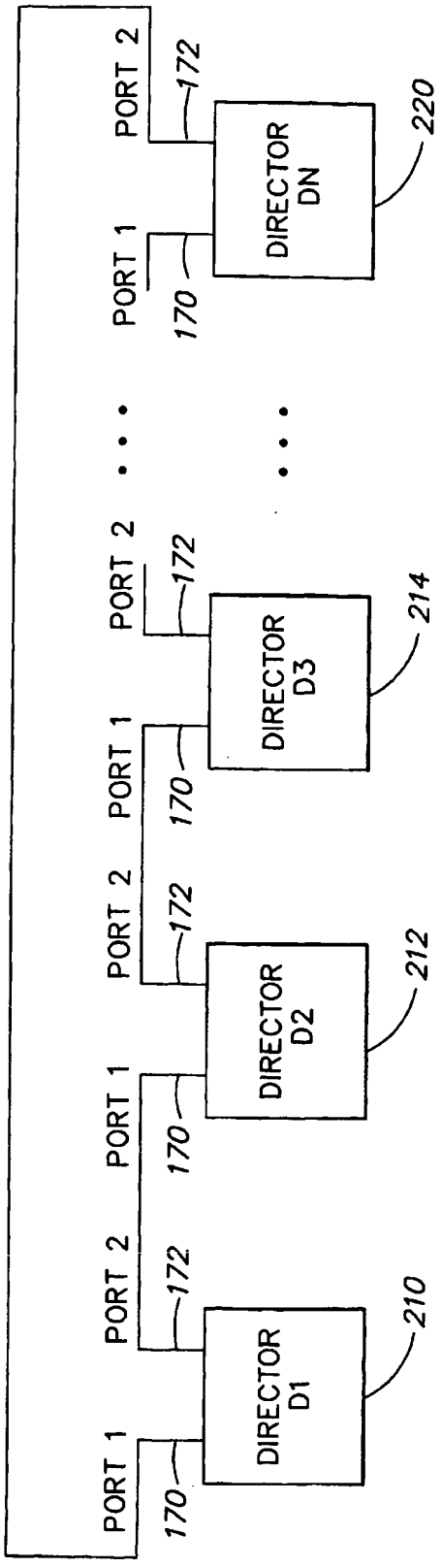

PORT 1
170
PORT 2
172
DIRECTOR D1
210
PORT 1
170
PORT 2
172
DIRECTOR D2
212
PORT 1
170
PORT 2
172
DIRECTOR D3
214
PORT 1
170
PORT 2
172
DIRECTOR DN
220

FIG. 3

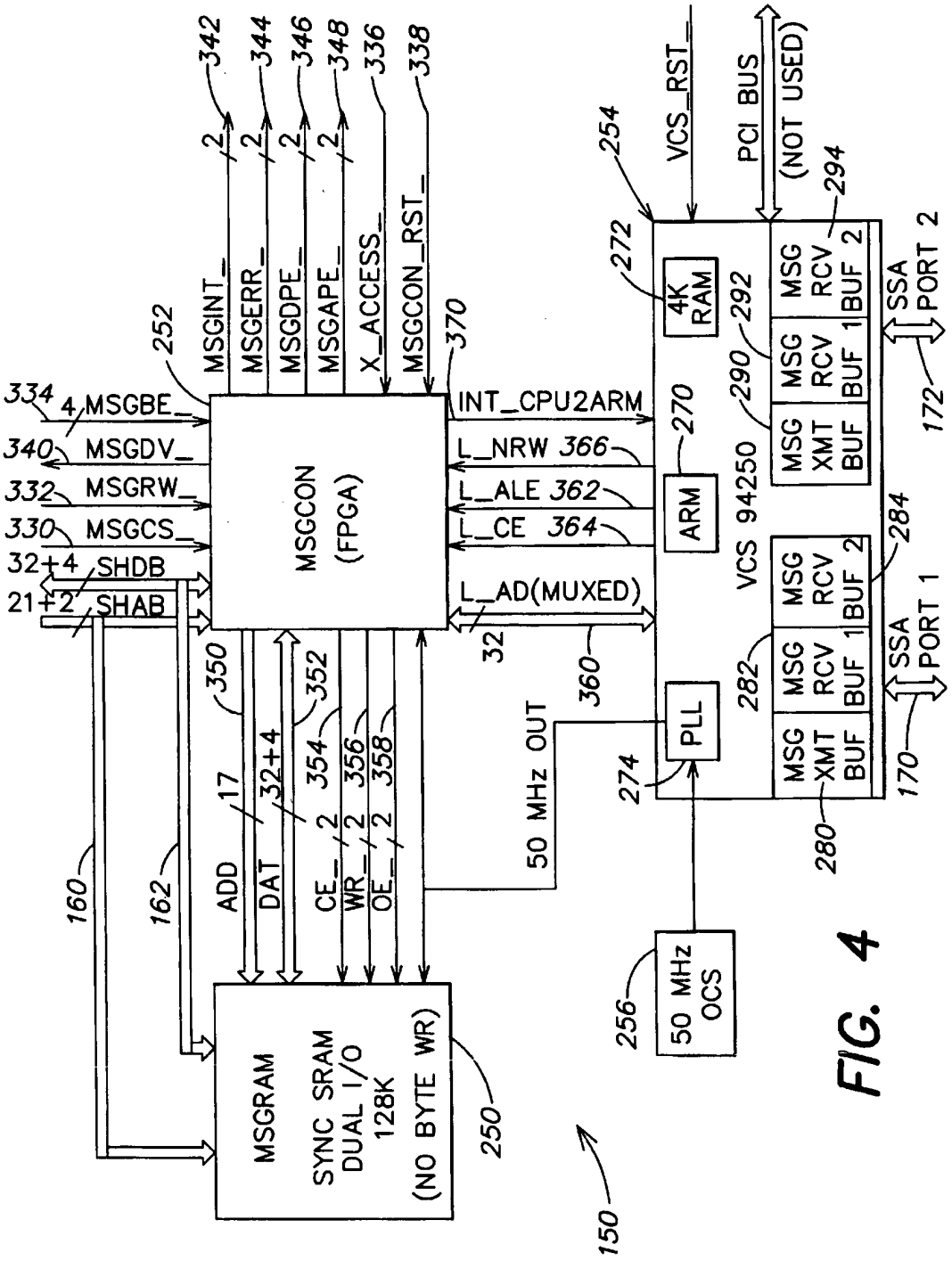

MSGCON (FPGA)
MSGRAM
SYNC SRAM DUAL I/O 128K (NO BYTE WR)
VCS 94250
ARM
4K RAM
PLL
50 MHz OCS
MSGINT_
MSGERR_
MSGDPE_
MSGAPE_
X_ACCESS_
MSGCON_RST_
MSGBE_
MSGDV_
MSGRW_
MSGCS_
SHDB
SHAB
INT_CPU2ARM
L_NRW 366
L_ALE 362
L_CE 364
L_AD(MUXED)
VCS_RST_
PCI BUS
(NOT USED)
SSA PORT 2
SSA PORT 1
MSG RCV BUF 1
MSG RCV BUF 2
MSG XMT BUF
ADD
DAT
CE_
WR_
OE_
50 MHz OUT
252
254
256
270
272
274
280
282
284
290
292
294
150
160
162
170
172
250
334
340
332
330
342
344
346
348
336
338
350
352
354
356
358
360
370

FIG. 4

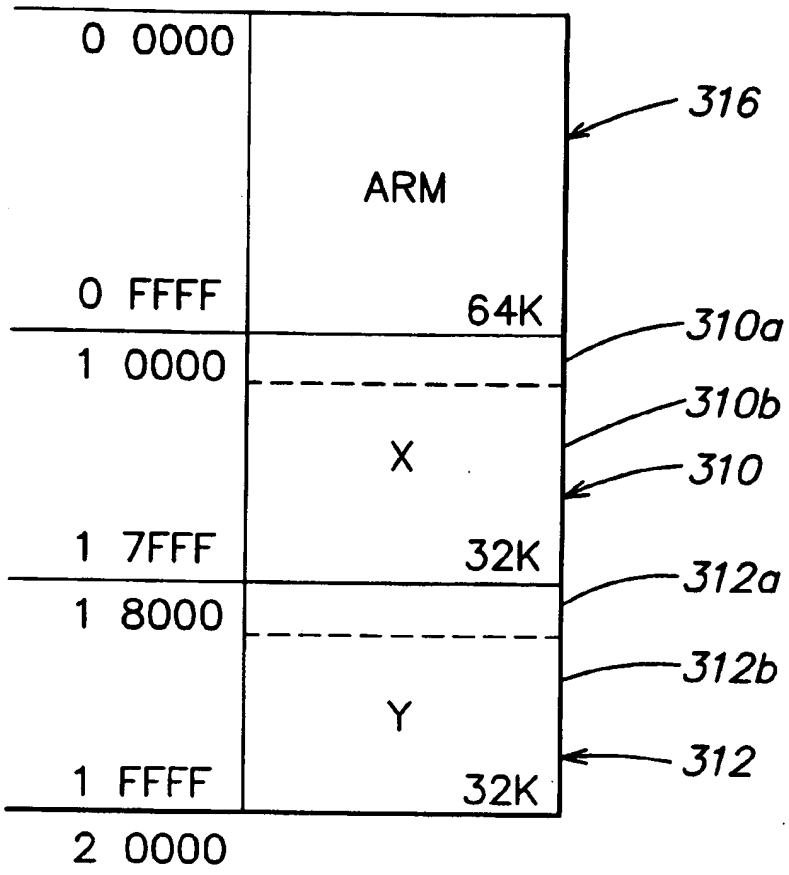

0 0000
ARM
0 FFFF
64K
316
310a
1 0000
310b
X
310
1 7FFF
32K
312a
1 8000
312b
Y
312
1 FFFF
32K
2 0000

FIG 5

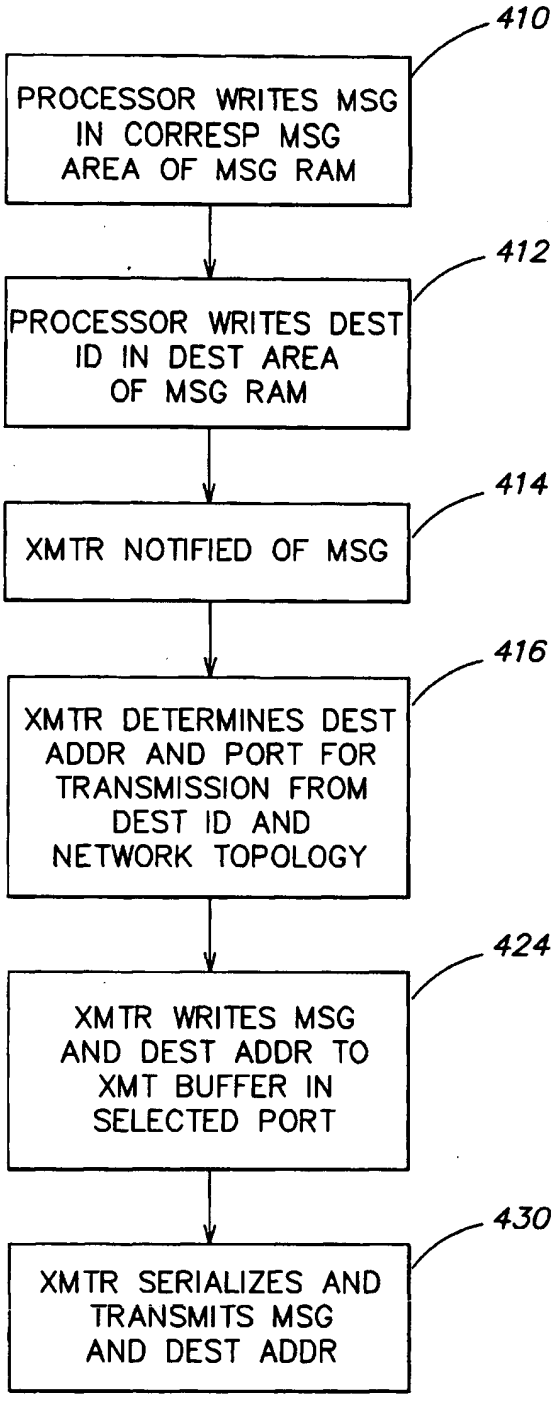

410
PROCESSOR WRITES MSG IN CORRESP MSG AREA OF MSG RAM
412
PROCESSOR WRITES DEST ID IN DEST AREA OF MSG RAM
414
XMTR NOTIFIED OF MSG
416
XMTR DETERMINES DEST ADDR AND PORT FOR TRANSMISSION FROM DEST ID AND NETWORK TOPOLOGY
424
XMTR WRITES MSG AND DEST ADDR TO XMT BUFFER IN SELECTED PORT
430
XMTR SERIALIZES AND TRANSMITS MSG AND DEST ADDR

FIG. 6

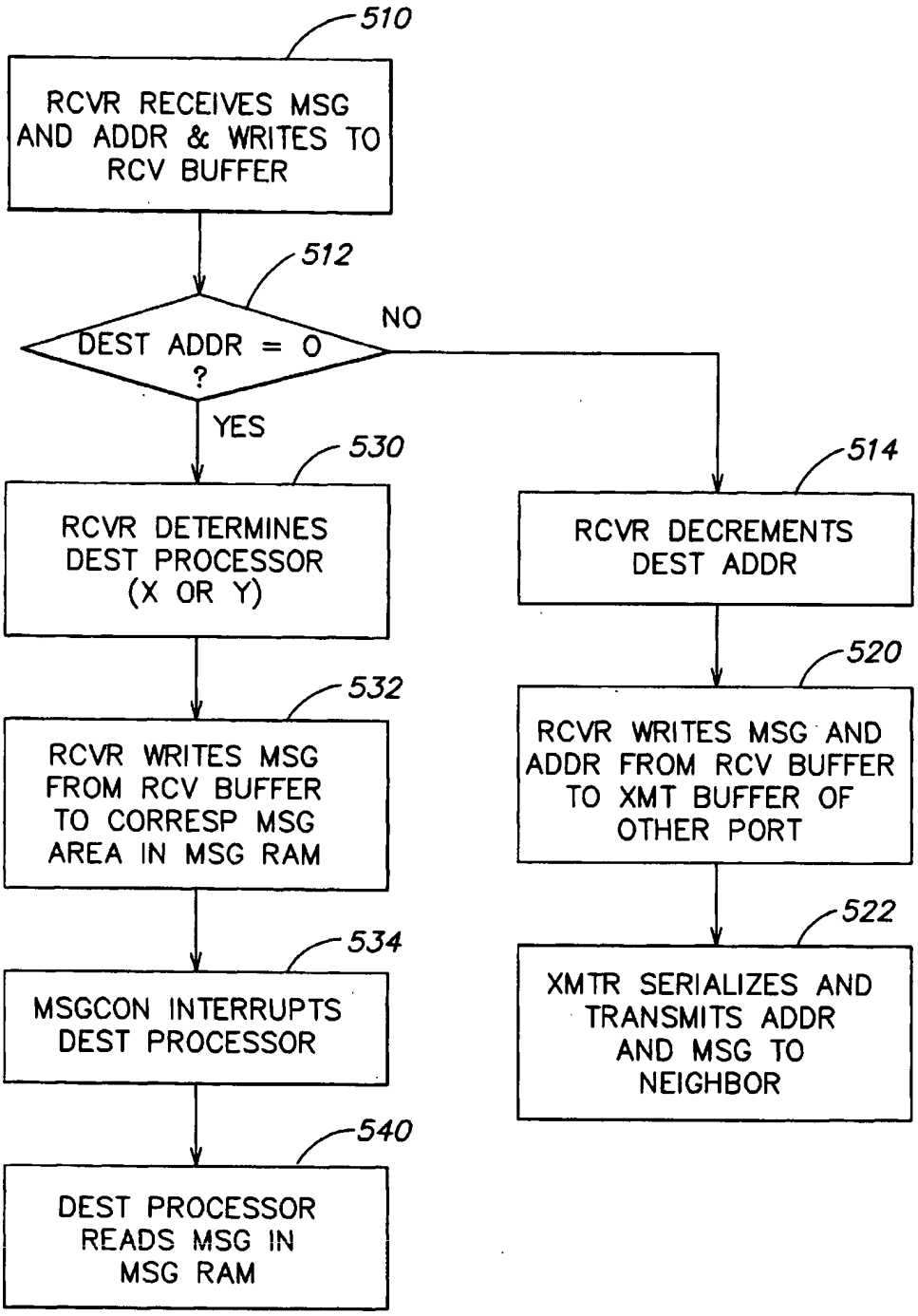

510
RCVR RECEIVES MSG AND ADDR & WRITES TO RCV BUFFER
512
DEST ADDR = 0 ?
NO
YES
530
RCVR DETERMINES DEST PROCESSOR (X OR Y)
532
RCVR WRITES MSG FROM RCV BUFFER TO CORRESP MSG AREA IN MSG RAM
534
MSGCON INTERRUPTS DEST PROCESSOR
540
DEST PROCESSOR READS MSG IN MSG RAM
514
RCVR DECREMENTS DEST ADDR
520
RCVR WRITES MSG AND ADDR FROM RCV BUFFER TO XMT BUFFER OF OTHER PORT
522
XMTR SERIALIZES AND TRANSMITS ADDR AND MSG TO NEIGHBOR

FIG. 7

METHODS AND APPARATUS FOR MESSAGE TRANSFER IN COMPUTER STORAGE SYSTEM

FIELD OF THE INVENTION

This invention relates to computer storage systems and, more particularly, to methods and apparatus for transferring messages between directors in disk array systems.

BACKGROUND OF THE INVENTION

Computer storage systems for high capacity, on-line applications are well known. Such systems use arrays of disk devices to provide a large storage capacity. To alleviate the delays inherent in accessing information in the disk array, a large capacity system cache memory is typically utilized. Controllers known as back end directors or disk adaptors control transfer of data from the system cache memory to the disk array and from the disk array to the system cache memory. Each back end director may control several disk devices, each typically comprising a hard disk drive. Controllers known as front end directors or host adaptors control transfer of data from the system cache memory to a host computer and from the host computer to the system cache memory. A system may include one or more front end directors and one or more back end directors.

The front end directors and the back end directors perform all functions associated with transfer of data between the host computer and the system cache memory and between the system cache memory and the disk array. The directors control cache read operations and execute replacement algorithms for replacing cache data in the event of a cache miss. The directors control writing of data from the cache to the disk array and may execute a prefetch algorithm for transferring data from the disk devices to the system cache memory in response to sequential data access patterns. The directors also execute diagnostic and maintenance routines. In general, the directors incorporate a high degree of intelligence.

During the operation of the computer storage system, it is frequently necessary to send messages between directors. For example, it may be necessary to transmit locking messages to other directors when a data block is being modified. In another example, a director may notify other directors of a hardware fault, so that appropriate steps may be taken. The mechanism used to transmit messages between directors is required to have low latency, because the receiving director may be required to act quickly or the transmitting director may require a quick response.

Prior art disk array systems have incorporated mechanisms for transferring messages between directors. In one prior art approach, a portion of the system cache memory functions as a mailbox for message transfer. A transmitting director writes a message in the mailbox section of system cache memory, and the receiving director reads the message. The messages are sent to and from system cache memory on the main data bus that is used for transfer of data between the system cache memory and the host computer or between the system cache memory and the disk array. This approach has disadvantages. First, the system cache memory and the buses connected to it are optimized for high data throughput rather than low latency. Accordingly, message transfer through the system cache memory may not provide sufficiently low latency. Furthermore, since messages are given priority over data transfers, data transfer performance is degraded as message traffic increases.

Another prior art approach to message transfer between directors involves an Ethernet local area network (LAN) that interconnects directors and is typically used for diagnostics, downloading of software, and the like. The LAN does not provide an acceptable level of latency in the transfer of messages and does not have inherent fault tolerant features to ensure data integrity. Accordingly, there is a need for improved methods and apparatus for message transfer in disk array systems.

SUMMARY OF THE INVENTION

According to a first aspect of the invention, a computer storage system is provided. The computer storage system comprises an array of storage devices, a system cache memory, one or more back end directors for controlling data transfer between the storage devices and the system cache memory, and one or more front end directors for controlling data transfer between the system cache memory and a host computer. At least two of the directors comprise a processor for controlling data transfer and a message interface for controlling high speed message transfer between the processors in the directors.

The message interface in each of the directors may comprise first and second ports for serial data transfer to other directors. Each of the first and second ports may be connected to another director to form a closed-loop network configuration. The message interface in each of the directors preferably utilizes the SSA interface protocol.

The message interface in each of the directors may comprise a transmit/receive circuit for transferring messages to and from other directors, a message memory for storing outgoing messages and incoming messages, and a message controller for controlling transfer of messages between the processor and message memory and between the message memory and the transmit/receive circuit.

In one embodiment, each director comprises a first processor and a second processor that share the message interface. An area of the message memory may be assigned to each of the first and second processors. Each processor area may include a message area for storing incoming and outgoing messages and a communication area for communication between the respective processor and the transmit/receive circuit.

Each processor may include means for writing an outgoing message in the message memory and means for notifying the message controller that the outgoing message has been written in the message memory. The message controller may include means for writing an incoming message in the message memory and means for notifying the respective processor that the incoming message has been written in the message memory.

According to another aspect of the invention, a method is provided for transferring messages between directors in a computer storage system. The computer storage system comprises an array of storage devices, a system cache memory, one or more back end directors for controlling data transfer between the array and the system cache memory, and one or more front end directors for controlling data transfer between the system cache memory and a host computer. Messages are transferred from a transmitting director to a receiving director on a high speed serial data link. As a result, the long latencies inherent in prior art message transfer techniques are overcome.

BRIEF DESCRIPTION OF THE DRAWINGS

For a better understanding of the present invention, reference is made to the accompanying drawings, which are incorporated herein by reference and in which:

FIG. 1 is a block diagram of a computer storage system suitable for incorporation of the invention;

FIG. 2 is a simplified block diagram of a director;

FIG. 3 is a block diagram showing the interconnection of front end directors and back end directors for message transfer in accordance with the invention;

FIG. 4 is a block diagram of an embodiment of the message interface shown in FIG. 2;

FIG. 5 is a schematic diagram that illustrates an example of the organization of the message memory shown in FIG. 4;

FIG. 6 is a flow chart that illustrates an example of a routine for transmitting messages in accordance with the invention; and

FIG. 7 is a flow chart that illustrates an example of a routine for receiving messages in accordance with the invention.

DETAILED DESCRIPTION

An example of a computer storage system suitable for incorporation of the present invention is shown in FIG. 1. A host computer **10** may be connected to the storage system using one or more channels or buses **12, 14, . . . 16**. The channels for communication with host computer **10** can be any suitable connection, such as a small computer system interface (SCSI), enterprise systems connection architecture (ESCON) or fiber channel (FC).

The storage system includes one or more front end directors **20, 22, . . . 24**, which are responsible for managing and translating read and write requests from host computer **10** into one or more requests corresponding to how data is stored on physical disk drives in the storage system. The front end directors **20, 22, . . . 24** are connected via buses **30** and **32** to a system cache memory **40**. The system cache memory **40** may be a random access memory having greater speed than the disk drives. If data being read is temporarily stored in the cache, a read request can be fulfilled more quickly by taking the data from system cache memory **40**. Similarly, when writing data, the data to be written can be stored in system cache memory **40**. System operation can proceed, while data is written from the system cache memory to the appropriate disk drive. The front end directors **20, 22, . . . 24** can be implemented in a number of ways, including a general purpose processor or a custom hardware implementation.

System cache memory **40** is coupled to disk drives **50, 52, . . . 54** through a back end director **60**. The storage system may include one or more back end directors, each connected to one or more disk drives. In the example of FIG. **1**, system cache memory **40** is coupled to disk drives **70, 72, . . . 74** through a back end director **62** and is coupled to disk drives **80, 82, . . . 84** through a back end director **64**. Each back end director **60, 62, . . . 64** may be implemented using a general purpose processor or a custom hardware implementation. Each back end director **60, 62, . . . 64** is connected to system cache memory **40** via buses **42** and **44**. Each of the buses **30, 32, 42** and **44** may be implemented, for example, as a 72 bit parallel bus. The system cache memory **40** may be a dual port random access memory. In one example, each back end director **60, 62, . . . 64** controls 4 disk drives, and the system may include up to 256 disk drives. An example of a computer storage system having the general configuration shown in FIG. **1** and described above is the Symmetrix model 5700, manufactured and sold by EMC Corporation.

A block diagram of an example of a suitable director architecture is shown in FIG. **2**. In one embodiment, the same architecture may be used for front end directors **20, 22, . . . 24** and back end directors **60, 62, . . . 64**. The director includes pipes **110** and **112**, each of which constitutes a high speed data path between the host computer **10** and system cache memory **40** in the case of a front end director or a high speed data path between the disk array and the system cache memory **40** in the case of a back end director. Pipes **110** and **112** are respectively connected to data buses **30** and **32** or to data buses **42** and **44**. Pipes **110** and **112** contain data transfer circuitry, the details of which are outside the scope of the present invention.

Pipes **110** are controlled by an X processor **120**, and pipes **112** are controlled by a Y processor **122**. The dual processor configuration provides high throughput and high efficiency in the operation of the computer memory system. The processors **120** and **122** include private resources required for high performance operation, such as local cache memory, a main memory, control circuitry and registers. X processor **120** is coupled to pipes **110** by a private address bus **124** and a private data bus **126**. Y processor **122** is coupled to pipes **112** by a private address bus **130** and a private data bus **132**.

The director also includes shared resources **140** and a message interface **150**. Processors **120** and **122**, shared resources **140** and message interface **150** are interconnected by a shared address bus **160** and a shared data bus **162**. Shared resources **140** includes those resources which are not critical to the performance of processors **120** and **122**. Shared resources **140** may include a variety of control functions, such as nonvolatile storage of software execution logs and error logs, and nonvolatile storage of software for processors **120** and **122**, and one or more connections to a local area network for diagnostic and maintenance purposes.

Message interface **150** is shared by processors **120** and **122** and provides a mechanism for high speed message transfer among directors. Messages may be transferred among front end directors, among back end directors and among front end and back end directors. A message is transferred from a source director to one or more destination directors. In the example of FIG. **2**, message interface **150** utilizes the SSA message protocol defined by ANSI document X3T10.1/0989D, "SSA Transport Layer One". Message interface **150** includes a first SSA port **170** and a second SSA port **172**. Each port **170, 172** provides bidirectional serial data transmission. Furthermore, the message interface **150** may be utilized for message transfer between processors on a single director. Thus for example, a message may be transferred from X processor **120** to Y processor **122** via message interface **150**, not using the SSA protocol or ports **170** and **172**.

A block diagram showing an interconnection between directors for message transfer in accordance with the invention is shown in FIG. **3**. In the example of FIG. **3**, the computer storage system includes N directors **210, 212, 214, . . . 220**, each of which may be a front end director or a back end director. A complete computer storage system requires at least one front end director and one back end director. Each director includes two SSA ports **170** and **172** and is connected to two logical neighbor directors in a closed-loop configuration. Thus, port **2** of director **210** is connected to port **1** of director **212**, and port **2** of director **212** is connected to port **1** of director **214**. Port **2** of director **220** is connected to port **1** of director **210**, thereby forming a closed-loop configuration. The closed-loop configuration provides redundancy in the operation of the message transfer

system. Thus, for example, if the connection between port 2 of director 210 and port 1 of director 212 is broken, a message may be transmitted from port 1 of director 210 to port 2 of director 212. The transmission of messages is described in more detail below.

A block diagram of an example of message interface 150 is shown in FIG. 4. Message interface 150 includes a message memory 250, a message controller 252, an SSA transmit/receive circuit 254 and an oscillator 256. In the example of FIG. 4, message memory 250 is a dual port random access memory (RAM) and may comprise a synchronous SRAM having a capacity of 128k bytes. The dual port feature permits separate access by the processors 120 and 122 (FIG. 2) and the transmit/receive circuit 254. Access to memory 250 is controlled through message controller 252.

Transmit/receive circuit 254 may include an Advanced Risc Machine (ARM) processor core 270, a memory 272, such as a 4k RAM, and a phase locked loop 274. Phase locked loop 274 receives the output of oscillator 256, typically at 50 MHz, and provides a 50 MHz output signal, synchronized to the device's external memory bus, to message memory 250 and message controller 252. In addition, transmit/receive circuit 254 may include a transmit buffer 280 and receive buffers 282 and 284 associated with port 170, and a transmit buffer 290 and receive buffers 292 and 294 associated with port 172.

Message interface 150 operates generally as follows. For transmission of messages, one of the processors 120 or 122 (FIG. 2) writes an outgoing message to message memory 250. The transmit/receive circuit 254 reads the outgoing message from the message memory 250, converts the message to serial format and transmits the message on one of the ports 170 and 172. When the message interface 150 receives a message, the message is examined to determine if it is addressed to that director or to a different director. When the message is addressed to another director, the message is transmitted through the other port to the next director in the closed-loop configuration. When the message is addressed to that director, the message is written to the message memory 250, and the appropriate processor is interrupted. The processor then reads the message from message memory 250. Transmission and reception of messages is described in more detail below.

A suitable organization of message memory 250 is shown in FIG. 5 for the example where the message memory 250 has a capacity of 128k bytes. Message memory 250 includes an X area 310 and a Y area 312, which are dedicated to the X processor 120 and the Y processor 122, respectively. Message memory 250 may also include an ARM area 316 which provides additional memory for ARM processor 270 in transmit/receive circuit 254 (FIG. 4). The areas 310 and 312 dedicated to the respective processors 120 and 122 may each be subdivided into a communication area 310*a*, 312*a* and a message area 310*b*, 312*b*. The message areas 310*b* and 312*b* may contain the data of messages being transmitted or received. The communication areas 310*a* and 312*a* permit communication between the respective processors and the ARM processor 270. Examples of information that may be placed in the communication areas 310*a* and 312*a* include destination identifiers of messages being transmitted, counts of messages in transmit and receive queues in the respective message areas 310*b* and 312*b*, and status information. In the example of FIG. 5, X and Y areas 310 and 312 have capacities of 32K bytes each, and ARM area 316 has a capacity 64K bytes. The message controller 252 may include write protect registers which ensure that X processor 120 accesses only X area 310 of message memory 250 and that Y processor 122 accesses only Y area 312.

As shown in FIG. 4, shared address bus 160 and shared data bus 162 are connected to message memory 250 and to message controller 252. As indicated above, shared address bus 160 and shared data bus 162 are connected to each of the processors 120 and 122 in the director. The processors provide control signals to the message controller 252, including a chip select signal 330 to select the message memory 250 or a register in the message controller 252, a read/write signal 332 to specify read or write, and a four-bit message byte enable signal 334 to enable a selected byte in the message. In addition, an X access signal 336 identifies the processor that is reading or writing a message, and a reset signal 338 resets the message controller.

Message controller 252 supplies signals to the respective processors 120 and 122, including a data valid signal 340, a message interrupt signal 342, a message error signal 344, a data parity error 346 and an address parity error 348. The message interrupt signal 342 and the error signals 344, 346 and 348 have two lines each, one corresponding to each of the processors 120 and 122. The message error signal 344 is reserved for errors other than parity errors. For example, the message error signal 344 may be used to indicate that one of the processors attempted to access an area of message memory 250 other than its assigned area.

Connections between message controller 252 and message memory 250 include an address bus 350 and a data bus 352. Shared address bus 160 and shared data bus 162 are connected to one port of message memory 250, and address bus 350 and data bus 352 are connected to the other port of message memory 250. Message controller 252 also provides control signals, including a chip enable signal 354, a write enable signal 356 and an output enable signal 358, to message memory 250. One control signal is provided for each of the two ports.

The connections between message controller 250 and transmit/receive circuit 354 include a multiplexed bus 360 which carries both address and data at different times. An address/data signal 362 indicates whether an address or data is present on bus 360. The transmit/receive circuit 254 also provides a chip enable signal 364 and a read/write signal 366 to message controller 252. Message controller 252 supplies an interrupt signal 370 to transmit/receive circuit 254 to indicate that a message has been written in message memory 250 and is awaiting transmission.

A flow chart that illustrates an example of a process for transmission of a message by one of the directors is shown in FIG. 6. In step 410, one of the processors 120 or 122 having a message to be transmitted writes the message in the corresponding area of message memory 250. Thus, assuming that Y processor 122 is transmitting a message, the message is written in message area 312*b* of Y area 312 (FIG. 5) in message memory 250. The processor places the address on shared address bus 160 and the message on shared data bus 162, and supplies a chip select signal 330 to message controller 252. Message controller 252 supplies a chip enable signal 354 and a write enable signal 356 to the appropriate port of message memory 250. This process transfers four bytes and is repeated until the entire message is transferred. The message may, for example, have a length of 32 bytes.

In step 412, the transmitting processor writes a message destination identifier (ID) in a destination area of message memory 250. The destination area may be in the communication area of message memory 250 that corresponds to the transmitting processor.

In step **414**, the transmitter in transmit/receive circuit **254** is notified of the message that was written to message memory **250**. In one approach, the communication area of message memory **250** for each processor has memory locations that contain a transmit queue and a receive queue. A value in the transmit queue may be incremented each time a message is written in memory by the corresponding processor and may be decremented when a message is transmitted. Similarly, a value in the receive queue may be incremented each time a received message is written in memory by the ARM processor **270** and may be decremented when a message is read by the corresponding processor. The transmit/receive circuit **254** may poll each transmit queue at specified intervals to determine whether there are messages waiting to be transmitted. A non-zero value in the transmit queue indicates messages waiting to be transmitted. In another approach, the message controller **252** may interrupt the transmit/receive circuit **254** to indicate that a message is awaiting transmission.

It will be understood that the ARM processor **270** in transmit/receive circuit **254** is programmed to perform both transmit and receive operations. The transmit/receive circuit is referred to as a transmitter when messages are being transmitted and is referred to as a receiver when messages are being received.

In step **416**, the transmitter in transmit/receive circuit **254** determines a destination address and a port for transmission of the message. The destination address and the port are determined from the destination ID and from knowledge of the network topology. The network topology specifies the location of each director in the message network. The network topology may, for example, be embodied in a table which associates each destination ID with a port that provides the shortest path to the destination ID and a destination address that corresponds to the destination ID. The destination address is a number representative of the number of nodes between the message transmitter and the message destination.

In step **424**, the transmitter writes the message from the message memory **250** to the transmit buffer in the selected port. When, for example, the message is transmitted through port **170**, the message and the destination address are placed in transmit buffer **280**. The transmit/receive circuit **254** communicates with message memory **250** via message controller **252**. In particular, transmit/receive circuit **254** requests writing of the message by asserting chip enable signal **364** to message controller **252**. Message controller **252** provides addresses on address bus **350**, chip enable signal **354** and output enable signal **358** to message memory **250**. The message is transferred via data bus **352** and multiplexed bus **360** to transmit buffer **280**.

In step **430**, the transmitter serializes and transmits the destination address and the message to the director connected to port **170** in accordance with the SSA protocol or other suitable serial data protocol. The message may be transmitted in a data packet that includes a header, a message body of up to 32 bytes and a trailer. The destination address may be part of the header.

The message transmission process shown in FIG. **6** and described above may be illustrated by way of example. With reference to FIG. **3**, assume that the Y processor in director **212** requires transmission of a message to the X processor **120** in director **220**. The Y processor in director **212** writes a destination ID, such as "220X", in the destination area of message memory **250**. The destination ID is used by the ARM processor **270** to access the table that contains the port and destination address of the X processor in director **220**. The shortest path from director **212** to director **220** is from port **1** of director **212** through director **210** to port **2** of director **220**. The table entry in director **212** which corresponds to director **220** thus specifies port **1** for transmission. The corresponding destination address has a value of 1 because there is one node between director **212** and director **220**. The message is thus transmitted by director **212** through port **1** with a destination address having a value of 1.

A flow chart of an example of a process for receiving messages in accordance with the invention is shown in FIG. **7**. In step **510**, the receiver in transmit/receive circuit **254** receives a serial message and a destination address and writes the message and the address to an available receive buffer. In the example described above, director **210** receives a message through port **172** from director **212**. The message and the destination address are written, for example, into receive buffer **294**. In step **512**, the receiver examines the received destination address. If the destination address is determined in step **512** to be nonzero, the message is not addressed to that director and must be transmitted to the next director in the network. In step **514**, the receiver decrements the received destination address by one. In step **520**, the receiver writes the message and the decremented address from the receive buffer to the transmit buffer in the other port of the message interface. In the above example, the message and the decremented address are written from receive buffer **294** in port **172** to transmit buffer **280** in port **170**. In step **522**, the transmitter serializes and transmits the decremented address and the message to the next director through the other port. In the example given above, the message is transmitted by director **210** to director **220** through port **170**.

If the received destination address is determined in step **512** to be zero, the received message is addressed to that director. In step **530**, the receiver determines the destination processor **120** or **122**. The destination processor in the director may, for example, be identified by a bit in the header of the received data packet. In step **532**, the receiver writes the received message from the receive buffer to the corresponding area in the message memory **250**. Assuming that the message has been received in receive buffer **294** and is addressed to the X processor **120** in the director, the message is written from receive buffer **294** to message area **310*b*** of X area **310**. The message is written to message memory via message controller **252**. In particular, the transmit/receive circuit **254** asserts the chip enable signal **364** and places the message on multiplexed bus **360**. The message is written to message memory **250** via address bus **350** and data bus **352**. The message controller **252** interrupts the destination processor in step **534** by asserting the X processor message interrupt line **342**. In step **540**, the destination processor (X processor **120**) reads the message in the message memory **250** via shared address bus **160** and shared data bus **162**.

The message interface **150**, in addition to handling high speed message transfer between directors, may handle message transfer between processors on the same director. Thus, for example, X processor **120** may transmit a message to Y processor **122** on the same director. In a manner similar to that described above, X processor **120** writes the message in message area **310*b*** of X area **310** and writes the destination ID of Y processor **122** in the destination area. When the transmitter examines the destination ID, it recognizes that transmission of the message to another director is not required. Instead, the message is written from X area **310** to Y area **312** of message memory **250**, and a message interrupt is sent to Y processor **122**. Y processor **122** then reads the

message in Y area **312**. Ports **170** and **172** are not used for transferring messages between processors on the same director.

The message interface **150** preferably includes error checking functions. The message controller **252** may check for address parity errors on shared address bus **160** during reads and writes and for data parity errors on shared data bus **162** during writes to message memory **250** and to message controller registers from the X and Y processors. These errors result in an address parity error signal **348** or a data parity error signal **346** to the appropriate processor. The message controller **252** may generate parity for processor reads from its registers and checks parity for processor reads from message memory **250**, but these errors do not cause an interrupt. A four-bit register in the message controller **252** may store which byte lane had errors on a CPU read. Another four-bit register may store the same information for processors writes, with a bit indicating that the access was to message memory **250** or a message controller register. Another register stores which address bytes had an address parity error. The multiplexed bus **360** has no parity. The message controller **252** generates parity on ARM processor writes to message memory **250** and checks parity on reads. If an error is detected, byte lane information is stored, and a message error signal **344** is sent to both processors.

While there have been shown and described what are at present considered the preferred embodiments of the present invention, it will be obvious to those skilled in the art that various changes and modifications may be made therein without departing from the scope of the invention as defined by the appended claims.

What is claimed is:

1. A computer storage system comprising:

an array of storage devices;

a system cache memory;

one or more back end directors for controlling data transfer between said array and said system cache memory; and

one or more front end directors for controlling data transfer between said system cache memory and a host computer, at least two of said directors comprising a processor for controlling data transfer and a message interface for controlling high speed message transfer between the processors in said at least two directors.

2. A computer storage system as defined in claim **1** wherein said message interface in each of said at least two directors comprises first and second ports for serial data transfer to others of said at least two directors.

3. A computer storage system as defined in claim **2** wherein each of said first and second ports is connected to another of said at least two directors and wherein said at least two directors are connected in a closed-loop network configuration.

4. A computer storage system as defined in claim **2** wherein the message interface in each of said at least two directors utilizes an SSA interface protocol.

5. A computer storage system as defined in claim **1** wherein the message interface in each of said at least two directors comprises a message memory for storing incoming messages and outgoing messages.

6. A computer storage system as defined in claim **1** wherein the message interface in each of said at least two directors transmits and receives messages on a high speed serial data link.

7. A computer storage system as defined in claim **1** wherein the message interface in each of said at least two directors comprises a transmit/receive circuit for transferring messages to and from others of said at least two directors, a message memory for storing outgoing messages and incoming messages and a message controller for controlling transfer of messages between said processor and said message memory and between said message memory and said transmit/receive circuit.

8. A computer storage system as defined in claim **7** wherein each of said at least two directors comprises a first processor and a second processor that share said message interface and wherein an area of said message memory is assigned to each of said first and second processors.

9. A computer storage system as defined in claim **7** wherein said processor includes means for writing an outgoing message in said message memory and said message controller includes means for notifying said transmit/receive circuit that said outgoing message has been written in said message memory.

10. A computer storage system as defined in claim **7** wherein said message controller includes means for writing an incoming message in said message memory and means for notifying said processor that said incoming message has been written in said message memory.

11. In a computer storage system comprising an array of storage devices, a system cache memory, one or more back end directors for controlling data transfer between said array and said system cache memory, and one or more front end directors for controlling data transfer between said system cache memory and a host computer, a method for transferring messages between said directors, comprising the step of:

transferring messages from a transmitting director to a receiving director on a high speed serial data link.

12. A method as defined in claim **11** wherein the step of transferring messages comprises the steps of writing an outgoing message in a message memory and notifying a transmitter that the outgoing message has been written in the message memory.

13. A method as defined in claim **11** wherein the step of transferring messages comprises writing an incoming message in a message memory and notifying a receiving processor that the incoming message has been written in the message memory.

14. A disk array system comprising:

an array of disk drives;

a system cache memory;

one or more back end directors for controlling data transfer between said array of disk drives and said system cache memory; and

one or more front end directors for controlling data transfer between said system cache memory and a host computer, each of said directors comprising a processor and a message interface for controlling high speed message transfer between the processors in said directors, the message interface in each of said directors comprising a transmit/receive circuit for transferring messages to and from others of said directors, a message memory for storing outgoing messages and incoming messages and a message controller for controlling the transfer of messages between said processor and said message memory and between said message memory and said transmit/receive circuit.

15. A disk array system as defined in claim **14** wherein the message interface in each of said directors further comprises

first and second ports for serial data transfer to others of said directors and wherein said directors are connected in a closed-loop configuration.

16. A disk array system as defined in claim 14 wherein each of said directors comprises a first processor and a second processor that share said message interface and wherein an area of said message memory is assigned to each of said first and second processors.

17. A disk array system as defined in claim 14 wherein said processor includes means for writing an outgoing message in said message memory and means for notifying said transmit/receive circuit that said outgoing message has been written in said message memory.

18. A disk array system as defined in claim 14 wherein said message controller includes means for writing an incoming message in said message memory and means for notifying said processor that said incoming message has been written in said message memory.

* * * * *

US006230229B1

(12) **United States Patent**
Van Krevelen et al.

(10) **Patent No.: US 6,230,229 B1**
(45) **Date of Patent: May 8, 2001**

(54) **METHOD AND SYSTEM FOR ARBITRATING PATH CONTENTION IN A CROSSBAR INTERCONNECT NETWORK**

(75) Inventors: **Christopher J. Van Krevelen**, Coon Rapids; **Reed S. Nelson**, Shoreview; **Don J. Hodapp, Jr.**, Maple Grove; **John D. Hamre**, Plymouth, all of MN (US)

(73) Assignee: **Storage Technology Corporation**, Louisville, CO (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **08/994,527**

(22) Filed: **Dec. 19, 1997**

(51) **Int. Cl.**[7] **G06F 13/00**

(52) **U.S. Cl.** **710/131**; 710/132; 710/129; 710/241; 710/113; 710/38; 710/39; 710/107

(58) **Field of Search** 395/306–312, 395/282–283, 284–285, 287–298, 840–848, 856–861; 710/240–244

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,897,833	*	1/1990	Kent et al.	370/85.2
4,985,830	*	1/1991	Atac et al.	710/131
5,307,466		4/1994	Chang .	
5,463,486		10/1995	Stevens .	
5,533,201	*	7/1996	Benton et al.	710/131
5,577,204	*	11/1996	Brewer et al.	395/200.01
5,623,698		4/1997	Stephenson et al. .	
5,657,449	*	8/1997	Osaki	370/357
5,682,485	*	10/1997	Farmer et al.	710/131
5,689,644	*	11/1997	Chou et al.	709/227
5,699,533	*	12/1997	Sakai	710/131
5,745,709	*	4/1998	Okabayashi et al.	395/311
5,751,710	*	5/1998	Crowther et al.	370/423
5,796,966	*	8/1998	Simcoe et al.	395/311
5,832,239	*	11/1998	Gavin et al.	395/285
5,835,739	*	11/1998	Bell et al.	710/128
5,838,937	*	11/1998	Lee et al.	395/311
5,854,906	*	12/1998	Van Loo	710/119
5,857,114	*	1/1999	Kim	395/842
5,859,975	*	1/1999	Brewer et al.	709/213
5,949,982	*	9/1999	Frankeny et al.	710/132
6,038,630	*	3/2000	Foster et al.	710/132

* cited by examiner

Primary Examiner—Robert Beausoleil
Assistant Examiner—Raymond N Phan
(74) *Attorney, Agent, or Firm*—Brooks & Kushman P.C.

(57) **ABSTRACT**

A method and system for transmitting data among a plurality of cards in a crossbar interconnect network having a plurality of cards each having source paths and destination paths utilizes a plurality of source arbitrators and a plurality of destination arbitrators each associated with the cards. The source arbitrators generate connection request commands from the source paths requesting access to a desired destination path and broadcasts the request for receipt by all of the destination arbitrators. The destination arbitrator associated with the desired destination path captures the connection request command and processes the command based on whether or not the desired destination path is busy. If the desired destination path is not busy, the destination arbitrator generates a connection command requesting a connection be made between the source path and the desired destination path. If the desired destination path is busy, the destination arbitrator stores the connection request command in one of a plurality of buffers until the desired destination path becomes available.

36 Claims, 3 Drawing Sheets

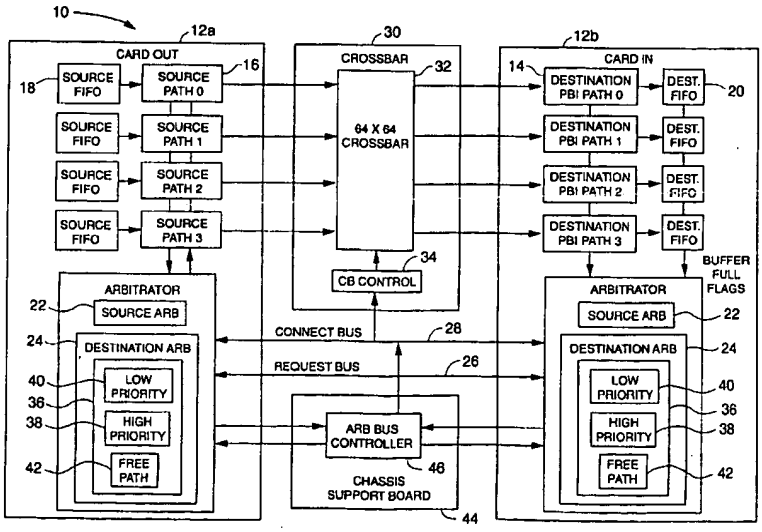
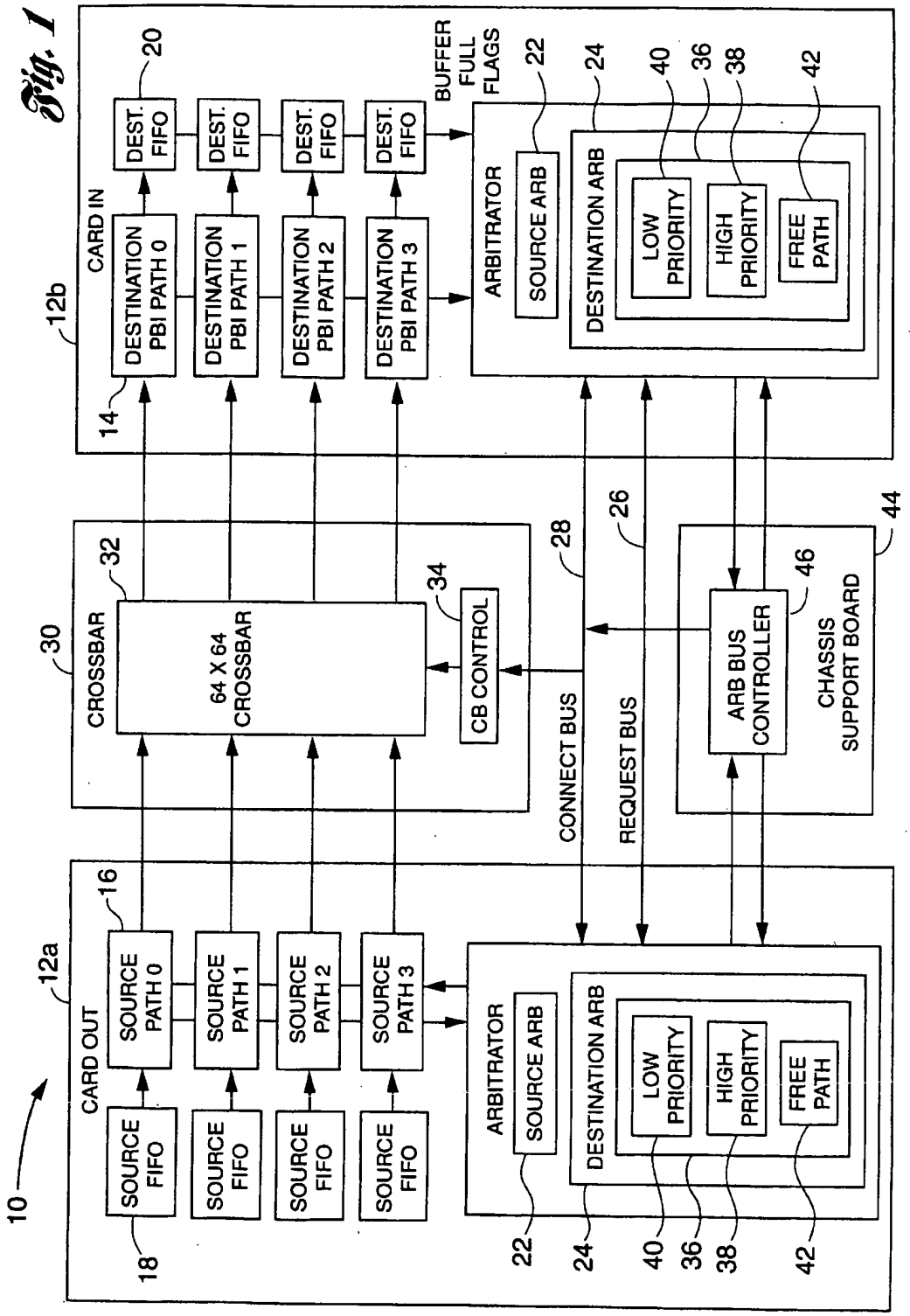


Fig. 1


10
12a
CARD OUT
16
SOURCE PATH 0
SOURCE PATH 1
SOURCE PATH 2
SOURCE PATH 3
SOURCE FIFO
18
ARBITRATOR
SOURCE ARB
DESTINATION ARB
LOW PRIORITY
HIGH PRIORITY
FREE PATH
22
24
40
36
38
42
30
CROSSBAR
32
64 X 64 CROSSBAR
34
CB CONTROL
28
CONNECT BUS
26
REQUEST BUS
44
CHASSIS SUPPORT BOARD
46
ARB BUS CONTROLLER
12b
CARD IN
14
DESTINATION PBI PATH 0
DESTINATION PBI PATH 1
DESTINATION PBI PATH 2
DESTINATION PBI PATH 3
20
DEST. FIFO
BUFFER FULL FLAGS

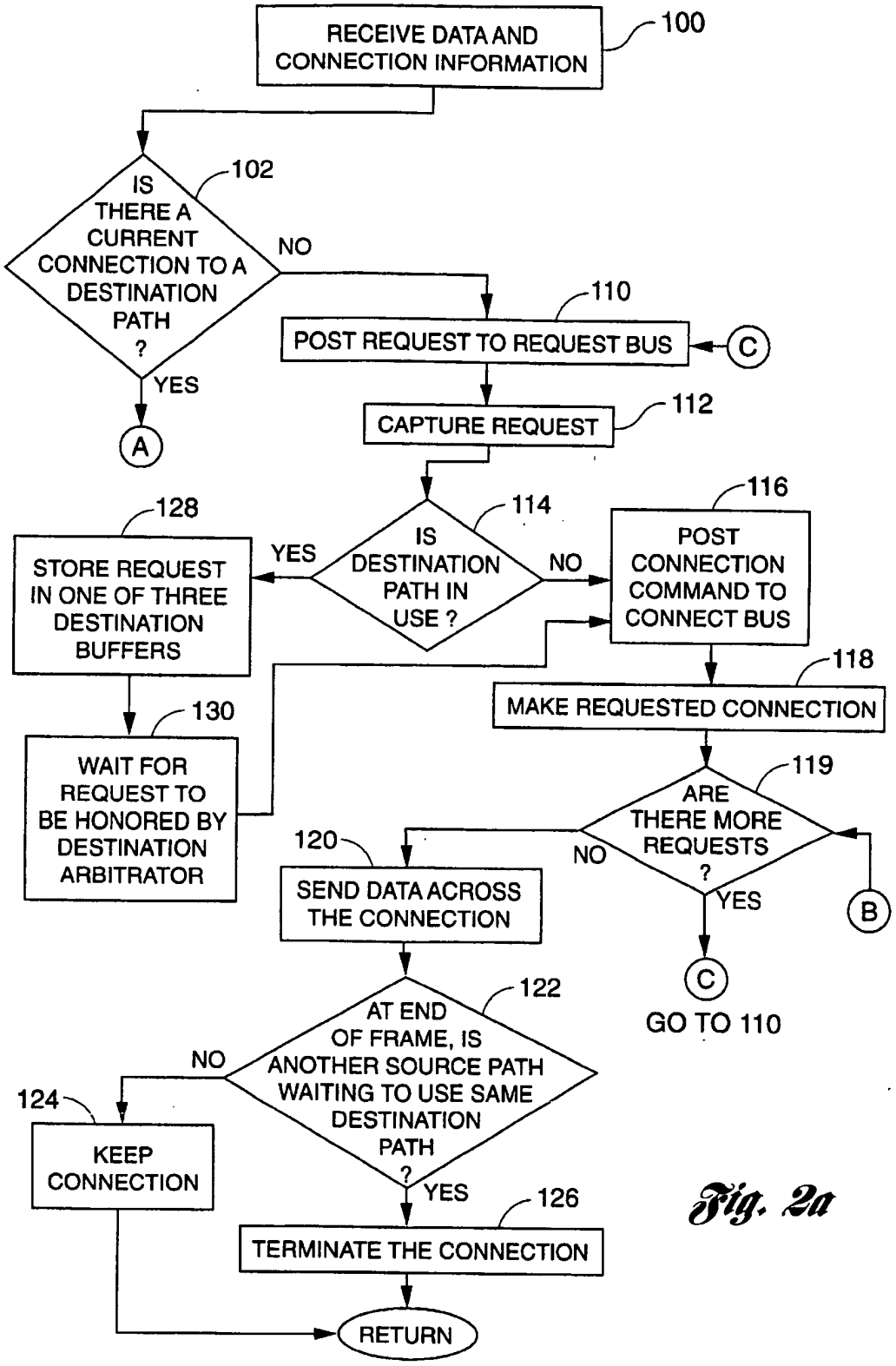

RECEIVE DATA AND CONNECTION INFORMATION
100
IS THERE A CURRENT CONNECTION TO A DESTINATION PATH ?
102
NO
YES
A
110
POST REQUEST TO REQUEST BUS
C
112
CAPTURE REQUEST
114
IS DESTINATION PATH IN USE ?
YES
NO
128
STORE REQUEST IN ONE OF THREE DESTINATION BUFFERS
130
WAIT FOR REQUEST TO BE HONORED BY DESTINATION ARBITRATOR
116
POST CONNECTION COMMAND TO CONNECT BUS
118
MAKE REQUESTED CONNECTION
119
ARE THERE MORE REQUESTS ?
NO
YES
B
C
GO TO 110
120
SEND DATA ACROSS THE CONNECTION
122
AT END OF FRAME, IS ANOTHER SOURCE PATH WAITING TO USE SAME DESTINATION PATH ?
NO
YES
124
KEEP CONNECTION
126
TERMINATE THE CONNECTION
RETURN

Fig. 2a

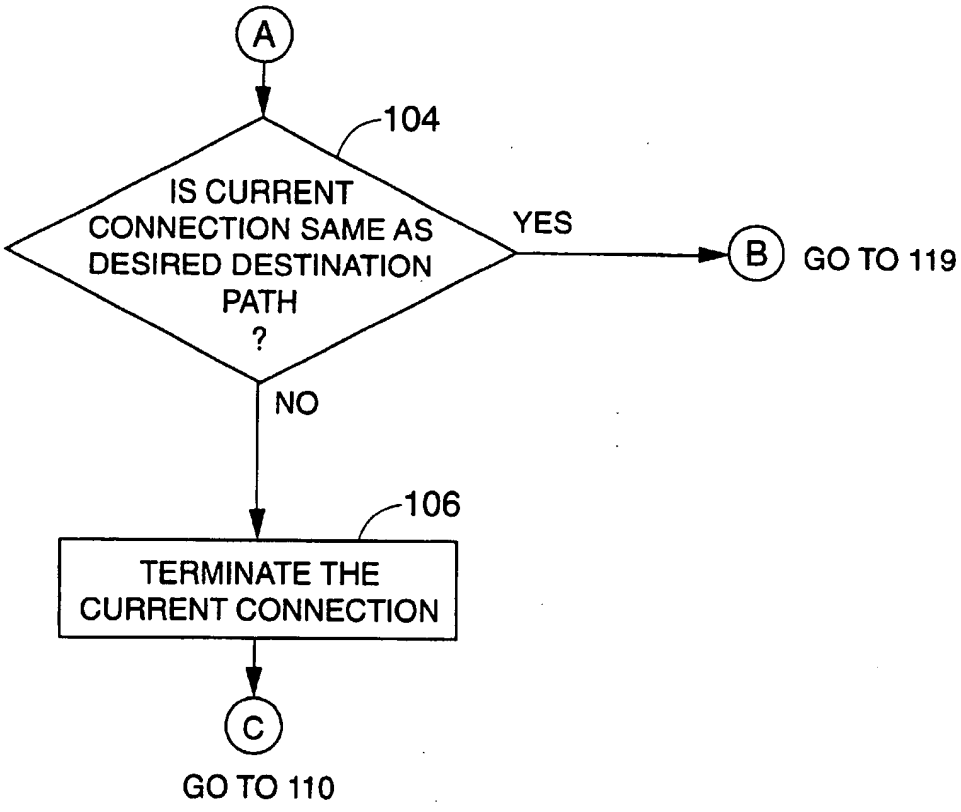

A
104
IS CURRENT CONNECTION SAME AS DESIRED DESTINATION PATH ?
YES
B
GO TO 119
NO
106
TERMINATE THE CURRENT CONNECTION
C
GO TO 110

Fig. 2b

METHOD AND SYSTEM FOR ARBITRATING PATH CONTENTION IN A CROSSBAR INTERCONNECT NETWORK

TECHNICAL FIELD

This invention relates to methods and systems for arbitrating path contention in an interconnect fabric utilizing serial or parallel crossbar switch technology.

BACKGROUND ART

Multiple cards having to communicate with each other have become widely used in computing systems. However, problems arise when the cards need to transfer data between each other and contend for communication path allocation.

In an interconnect fabric using serial or parallel crossbar switch technology, path arbitration is required when path contention occurs. For example, in a system having an NxN fully connected non-blocking crossbar, all of the cards in the system have their input paths fully connected to the crossbar. A problem arises when more than one source path needs to send data to the same destination path at the same time. A source path that is connected to a destination path may remain connected for a long period of time blocking all traffic intended for that path regardless of priority. The source path(s) end up continuously trying to gain access to the destination path until it becomes available without any assurance it will ever get connected to the desired destination path. This is referred to as a lock-out condition when one source path cannot make its connection. A fair system of arbitration would capture the connection requests and honor them in the order in which they were received.

Thus, there exists a need for path arbitration in such a system when path contention occurs, and to process high priority requests ahead of low priority requests.

DISCLOSURE OF THE INVENTION

It is a general object of the present invention to provide a method and system for allocating paths in a crossbar interconnect network.

In carrying out the above object and other objects, features, and advantages of the present invention, a method is provided for transmitting data among a plurality of cards in a crossbar interconnect network, each of the plurality of cards having source paths for originating the data and destination paths for receiving the data. The method includes the step of generating a connection request command from one of the source paths requesting access to a desired one of the destination paths. The method also includes the step of capturing the connection request command at the desired destination path. Still further, the method includes the step of processing the connection request command based on whether or not the desired destination path is busy so as to prevent a lock-out condition and to fairly allocate the desired destination path.

In further carrying out the above object and other objects, features, and advantages of the present invention, a system is also provided for carrying out the steps of the above described method. The system includes a source arbitrator associated with each of the cards and in communication with each of the source paths of the associated card for generating a connection request command from one of the source paths requesting access to a desired one of the destination paths. The system also includes a destination arbitrator associated with each of the cards for capturing the connection request command at the desired destination path and processing the request command based on whether or not the desired destination path is busy.

The above object and other objects, features and advantages of the present invention are readily apparent from the following detailed description of the best mode for carrying out the invention when taken in connection with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** is a schematic block diagram of an interconnect network; and

FIGS. **2***a* and **2***b* are a flow diagram illustrating the general sequence of steps associated with the present invention.

BEST MODES FOR CARRYING OUT THE INVENTION

FIG. **1** is a schematic block diagram of an interconnect network system in which the present invention may be employed, denoted generally by reference numeral **10**. The system **10** includes a plurality of cards **12**, two of which are shown as **12***a*, **12***b*. Each of the cards **12** have N input/destination paths **14** and M output/source paths **16** for communicating with each of the other cards. For example, the system **10** may include sixteen cards each having four inputs and four outputs. However, the number of inputs does not have to equal the number of outputs. Furthermore, for ease of illustration, only the source paths **16** are shown for card **12***a* while only the destination paths **14** are shown for card **12***b*. In actuality, card **12***a* would have similar destination paths **14** as card **12***b*, and card **12***b* would have similar source paths **16** as card **12***a*.

Each of the source paths **16** have a source buffer **18** associated therewith. Each source buffer **18** stores data and connection information for receipt by its respective source path **16**. A frame of data is stored in the source buffer **18** in which the first line of the frame instructs the source path **16** what destination path the data should be sent to and what type of connection to make, i.e., high priority, low priority, or free path. A high or low priority request type is a request to talk to a specific path on the requested destination card. A free path request type is a request to talk to any available path on the requested destination card.

Similarly, each of the destination paths **14** has a destination buffer **20** associated therewith for storing the data received by its respective destination path **14** until retrieved for subsequent processing or forwarding. Preferably, source buffer **18** and destination buffer **20** each have a first-in, first-out (FIFO) queue structure.

Each card **12** also includes a source arbitrator **22** and a destination arbitrator **24**. Source arbitrator **22** is in communication with each of the source paths **16** for receiving connection requests. Upon reading the frame of data stored in the source buffer **18**, source path **16** determines which destination path **14** the data needs to be sent to and initiates a connection request command to the source arbitrator **22**. Each connection request command is then transferred to a request bus **26** by the source arbitrator **22** to be captured by the destination arbitrator **24** responsible for the requested destination path. Preferably, each source arbitrator **22** transfers only one request to any one destination path **14** at a time. That is, if more than one source path **16** on its card wants to talk to the same destination path **14**, source arbitrator **22** will transfer one request and hold the rest. Once the transferred request is completed, source arbitrator **22** will then transfer

the next request to the same destination path **14**. This is done so as to avoid a 64-deep FIFO buffer associated with each destination path as described in greater detail below.

Each of the destination arbitrators **24** is in communication with their associated destination paths **14** and destination buffers **20**. Destination arbitrator **24** captures each of the connection request commands for the destination paths **14** on its board. If the requested destination path **14** is not in use, destination arbitrator **24** will post a connection command to a connect bus **28**. Furthermore, the destination arbitrator **24** is in communication with each of the destination buffers **20** so that when any of the destination buffers **20** become almost full, destination arbitrator **24** will prevent data from being sent until instructed otherwise as described below. Upon noticing the destination buffer **20** becoming almost full, the destination arbitrator **24** sends a "buffer almost full" command across the connect bus **28** to the source arbitrator **22**. The source arbitrator **22** will then instruct the source path **16** to stop sending data until a "resume command" is sent by the destination arbitrator **24** when the destination buffer **20** is no longer almost full.

Connections between source paths **16** and destination paths **14** are accomplished via a crossbar card **30**. Crossbar card **30** includes a switch **32** having 16Xn inputs and 16Xm outputs (where "16" represents the number of cards in the system) capable of connecting each of the source paths **16** to one of the destination paths **14** so as to transfer data between each of the cards **12**. Crossbar card **30** also includes a crossbar (CB) control **34** for monitoring the connect bus **28** and instructing the switch **32** as to which source path **16** should be connected to a particular destination path **14**. It should be noted that all of the destination paths **14** can be connected at the same time if all of the source paths **16** need to send data to different destination paths. The present invention also supports broadcasting, wherein one source path **16** can send data to one or more or all destination paths **14** at one time.

If the requested destination path **14** is in use, destination arbitrator **24** puts the request in a request buffer **36** associated with the requested destination path **14**. That is, even though there is only one request buffer **36** shown in FIG. **1**, there is actually one request buffer **36** for each destination path **14** for each card **12**. Preferably, request buffer **36** is a 16-deep first-in, first-out (FIFO) buffer. In addition, each request buffer **36** preferably includes three sets of buffers for storing three types of requests, i.e., a low priority request buffer **38**, a high priority request buffer **40** and a free path request buffer **42**. When the requested destination path becomes available, destination arbitrator **24** will post the connection request command stored in the request buffer **36** to the connect bus **28** for receipt by CB control **34**. Since the source arbitrator **22** preferably sends at the most only one of each request type to any one destination path **14** at a time, the request buffer **36** only has to keep track of **16** requests of each type, one from each card, not 16Xn or 64 (one from each source path). However, if the request buffer **36** is 64-deep, the request buffer **36** would be able to keep track of all the possible requests from all the cards **12** so as to not only prevent lock-out, but to also make arbitration totally fair. That is, if the request buffer **36** is only 16-deep, then the source path **16** can only send one connection request command for the desired destination path, while another source path **16** on a different card **12** can send a connection request command for the desired destination path, even though earlier requests are still waiting to be connected.

A request bus **26** and a connect bus **28** interconnect each of the source arbitrators **22** with each of the destination arbitrators **24**. Although a separate request bus **26** and connect bus **28** are disclosed herein, the present invention can employ a single bus supporting both types of commands if desired.

A chassis support board **44** includes an arbitrator bus controller **46** in communication with each of the source arbitrators **22** and destination arbitrators **24** for allocating the request and connect buses **26,28**, respectively, to ensure only one of the arbitrators **22,24** drives the buses **26,28** at a time. That is, arbitrator bus controller **46** instructs the cards **12** when they can use the request bus **26** or the connect bus **28** after any one of the arbitrators **22,24** requests use of one of these buses.

Turning now to FIGS. **2***a* and **2***b*, there is shown a flow diagram illustrating the general sequence of steps associated with the present invention. First, the source path **16** receives a frame of data and connection information from its associated source buffer **18**, as shown at block **100**. Next, a determination is made as to whether or not the source path **16** has a current connection to any destination path **14**, as shown at conditional block **102**. If not, the source arbitrator **22** posts a connection request command to the request bus **26**, as shown at block **110**. The destination card **12** having the requested destination path **14** captures the request, as shown at block **112**.

If the destination path **14** is not in use, destination arbitrator **24** will post a connection command to the connect bus **28**, as shown at conditional block **114** and block **116**. The CB control **34** receives the command and instructs switch **32** to connect the source path **16** to the requested destination path **14**, as shown at block **118**. The source arbitrator **22**, which is constantly monitoring the connect bus **28**, will recognize its connection request being sent across the connect bus **28** and know when it can then start sending data.

Before sending the data, however, a determination is made as to whether or not the data needs to be sent to additional connections, as shown at conditional block **119**. This is known as multi-casting wherein the source path **16** can talk to two or more destination paths **14** at once. If there are no more connection requests, the source path **16** then begins sending data across the connection, as shown at block **120**. If, on the other hand, there are more connection requests, the source arbitrator **22** will return to block **110** and post another request to the request bus **26**. Thus, the source path **16** will request connections one at a time until all connection are made before the data is sent at once to all the destination paths **14**.

Data is broken up into frames of a maximum size. A connection between a source path **16** and a destination path **14** is typically made for one frame at a time. When an end of frame is encountered, a determination is made as to whether or not there is another path waiting to use the same destination path, as shown at conditional block **122**. Again, this determination is made by the source arbitrator **22** continuously monitoring the request bus **26** to identify if any requests are made for the current destination path. If not, the end of the frame is sent with an instruction to keep the connection, block **124**, so that another frame can be sent if the next frame is going to the same destination path or just to hold the connection until another source path **16** needs access to the same destination path.

However, if there is another source path **16** waiting to use the same destination path **14**, an end of frame terminated (EOFT) command is sent with an instruction to terminate the connection, as shown at block **126**. Thus, the destination path **14** will become available for the next request.

Returning to conditional block **102**, if there is a current connection between the source path **16** and a destination path **14**, as might be the case as described above at blocks **122** and **124**, the method proceeds to determine if the connection is with the desired destination path, as shown at conditional block **104**. If so, the source path **16** immediately proceeds to send the data, as shown at block **120**. If not, the connection is terminated, and the source arbitrator **22** proceeds to post the connection request command, as shown at block **110**.

Returning to conditional block **114**, if the destination path is already in use, destination arbitrator **24** will store the request in the request buffer **36**, as shown at block **128**. As described above, destination arbitrator **24** will then store the request in one of the three buffers **36** associated with the requested destination path **14**, as shown at block **128**. At this point, the request is stored and waits to be honored by the destination arbitrator **24**, as shown at block **130**.

When the destination path **14** becomes available, destination arbitrator **24** will first check to see if there is a high priority request stored in the high priority FIFO **38**. If so, the method proceeds to block **116** to make the connection. If not, the method proceeds to determine if there is a low priority request stored in the low priority request FIFO **40**. If so, the method proceeds to block **116** to make the connection.

If there is neither a high priority request or a low priority request stored in request buffer **36**, the method proceeds to determine if there is a free path request for the destination path **14** stored in the free path request FIFO **42**. If so, the method proceeds to block **116** to make the connection. Otherwise, the request remains stored until the desired destination path becomes available.

While the best modes for carrying out the invention have been described in detail, those familiar with the art to which this invention relates will recognize various alternative designs and embodiments for practicing the invention as defined by the following claims.

What is claimed is:

1. A method for transmitting data among a plurality of cards in a crossbar interconnect network, each of the plurality of cards having source paths for originating the data and destination paths for receiving the data, the method comprising:

generating a connection request command from one of the source paths requesting access to a desired one of the destination paths;

capturing the connection request command at the desired destination path;

generating a connection command from the desired destination path for connecting the one of the source paths to the desired destination path if the desired destination path is not busy;

storing the connection request command from each of the source paths associated with each of the plurality of cards at the desired destination path if the desired destination path is busy;

continuously monitoring the desired destination path to determine when the desired destination path is no longer busy; and

generating a second connection command from the desired destination path based on the connection request command when the desired destination path is no longer busy.

2. The method as recited in claim **1** wherein generating the connection command further includes transmitting the data from the one of the source paths to the desired destination path.

3. The method as recited in claim **1** wherein generating the second connection command includes determining a priority of the connection request command stored in the request buffer.

4. The method as recited in claim **2** wherein the method further comprises:

determining if any additional connection requests relating to the data exist prior to transmitting the data; and

if so, generating corresponding connection request commands from the one of the source paths requesting access to additional destination paths.

5. The method as recited in claim **1** wherein the connection request command is stored in a first-in first-out manner.

6. The method as recited in claim **2** wherein transmitting the data includes:

determining if the desired destination path is almost full; and

if so, generating a pause command for receipt by the one of the source paths instructing the one of the source paths to stop transmitting the data.

7. The method as recited in claim **2** wherein transmitting the data further includes:

determining if a second one of the source paths is requesting access to the desired destination path upon completion of transmitting the data; and

if so, terminating the connection between the one of the source paths and the desired destination path so as to allow the second one of the source paths access to the desired destination path.

8. The method as recited in claim **6** further comprising generating a resume command for receipt by the one of the source paths when the desired destination path is no longer almost full, the resume command instructing the one of the source paths to resume transmitting the data.

9. The method as recited in claim **7** wherein the step of determining if a second one of the source paths is requesting access to the desired destination path occurs when the end of a frame of data is encountered.

10. The method as recited in claim **7** further comprising holding the connection between the one of the source paths and the desired destination path if it is determined that no other source paths are requesting access to the desired destination path.

11. A system for transmitting data among a plurality of cards in a crossbar interconnect network, each of the plurality of cards having source paths for originating the data and destination paths for receiving the data, the system comprising:

a source arbitrator associated with each of the cards and in communication with each of the source paths of the associated card for generating a connection request command from one of the source paths requesting access to a desired destination path;

a destination arbitrator associated with each of the cards and in communication with each of the destination paths for capturing the connection request command at the desired destination path, processing the connection request command based on whether or not the desired destination path is busy, and continuously monitoring the desired destination path to determine when the desired destination path is no longer busy; and

a plurality of request buffers corresponding to each of the destination paths, wherein the destination arbitrator, in processing the connection request command, stores the connection request command in the request buffer corresponding to the desired destination path if the desired destination path is busy.

12. The system as recited in claim 11 further comprising a switch in communication with each of the destination arbitrators and wherein the destination arbitrator, in processing the connection request command, is further operative to generate a connection command for receipt by the switch if the desired destination path is not busy, and wherein the switch connects the one of the source paths to the desired destination path in response to the connection command so as to allow the one of the source paths to transmit the data.

13. The system as recited in claim 11 wherein each of the request buffers has a depth large enough to store the connection request commands from each of the source paths associated with each of the plurality of cards so as to fairly allocate the desired destination path.

14. The system as recited in claim 11 wherein the destination arbitrator, in continuously monitoring the desired destination path, is further operative to generate a second connection command based on the connection request command stored in the request buffer associated with the desired destination path when the desired destination path is no longer busy.

15. The system as recited in claim 11 wherein each of the request buffers is a first-in first-out buffer.

16. The system as recited in claim 12 wherein the source arbitrator generates additional connection request commands from the one of the source paths requesting access to additional desired destination paths prior to transmitting the data.

17. The system as recited in claim 12 further comprising a destination buffer corresponding to each of the destination paths and wherein the destination arbitrator is further operative to determine if the destination buffer associated with the desired destination path is almost full and, if so, generate a pause command for receipt by the one of the source paths instructing the one of the source paths to stop transmitting the data.

18. The system as recited in claim 14 wherein the destination arbitrator, in generating the second connection command is further operative to determine a priority of the connection request command stored in the request buffer associated with the desired destination path.

19. The system as recited in claim 12 wherein the source arbitrator is further operative to determine if a second one of the source paths is requesting access to the desired destination path after transmission of the data and to generate a connection termination command if a second one of the source paths is requesting access to the desired destination path.

20. The system as recited in claim 17 wherein the destination arbitrator is further operative to generate a pause command for receipt by the one of the source paths when the destination buffer is no longer almost full, the resume command instructing the one of the source paths to resume transmitting the data.

21. The system as recited in claim 18 wherein each of the request buffers include a high priority request buffer for storing high priority requests.

22. The system as recited in claim 18 wherein each of the request buffers include a low priority request buffer for storing low priority requests.

23. The system as recited in claim 18 wherein each of the request buffers include a free path request buffer for storing free path requests requesting access to any available destination path on a particular card.

24. A scaleable apparatus for transmitting data among a plurality of cards in a crossbar interconnect network, each of the plurality of cards having source paths for originating the data and destination paths for receiving the data, the apparatus comprising:

each of the cards including a source arbitrator in communication with each of the source paths and a destination arbitrator in communication with each of the destination paths, each of the destination arbitrators including a request buffer for each of the destination paths, the source arbitrators generating a connection request command for each of the source paths requesting access to a desired one of the destination paths, and the destination arbitrators capturing the connection request commands directed to any of the destination paths associated with its respective card, storing the connection request commands in the request buffer associated with the desired destination path if the desired destination path is busy, continuously monitoring the desired destination path to determine when the desired destination path is no longer busy, and generating connection commands when the desired destination path is not busy so that data may be transmitted;

a communication bus coupling each of the source arbitrators with each of the destination arbitrators, the communication bus for receiving the connection request commands from each of the source arbitrators and broadcasting the connection request commands for receipt by each of the destination arbitrators and for receiving and broadcasting the connection commands from each of the destination arbitrators; and

a switch connected to each of the source paths and each of the destination paths and the communication bus for connecting each of the source paths with a corresponding desired destination path upon receipt of a corresponding connection command.

25. The apparatus as recited in claim 24 wherein the communication bus includes:

a request bus for receiving and broadcasting the connection request commands; and

a connect bus for receiving and broadcasting the connection commands.

26. The apparatus as recited in claim 24 further comprising:

an arbitration bus controller coupled to each of the source arbitrators and each of the destination arbitrators for determining when each of the source arbitrators and each of the destination arbitrators may transmit a connection request command and a connection command, respectively.

27. The apparatus as recited in claim 24 wherein each of the request buffers has a depth large enough to store the connection request commands from each of the source paths associated with each of the plurality of cards so as to fairly allocate the desired destination path.

28. The apparatus as recited in claim 24 wherein each of the request buffers include priority buffers to determine a priority of the stored connection command associated with the desired destination buffer.

29. The apparatus as recited in claim 24 wherein the source arbitrators generate additional connection request commands for each of the source paths requesting access to additional destination paths prior to transmitting the data.

30. The apparatus as recited in claim 24 wherein each of the destination paths include a destination buffer and wherein each of the destination arbitrators are operative to determine if the destination buffer associated with the desired destination path is almost full and, if so, generate a pause command by receipt by the source path transmitting the data instructing the source path to stop transmitting data.

31. The apparatus as recited in claim **24** wherein each of the source arbitrators is further operative to determine if another one of the source paths is requesting access to the desired destination path based on the connection request commands broadcast by the communication bus and, if so, to generate a connection termination command after transmitting the data.

32. The apparatus as recited in claim **28** wherein the priority buffers include a high priority request buffer for storing high priority requests.

33. The apparatus as recited in claim **28** wherein the priority buffers include a low priority request buffer for storing low priority requests.

34. The apparatus as recited in claim **28** wherein the priority buffers include a free path request buffer for storing free path requests requesting access to any available destination path.

35. The apparatus as recited in claim **28** wherein each of the request buffers is a first-in first-out buffer.

36. The apparatus as recited in claim **30** wherein each of the destination arbitrators are further operative to transmit a resume command to the source path upon the destination buffer no longer being almost full.

* * * * *